# 尚马教育 JAVA 基础课程

# Lang包常用类

文档编号：A11

创建日期： 2017-04-12

最后修改日期：2019-09-23

版 本 号：V3.0

电子版文件名：尚马教育-第一阶段-11.jdk语言包专题课程.docx

**文档修改记录：**

|  |  |  |  |
| --- | --- | --- | --- |
| 更新日期 | 更新作者 | 更新说明 | 版本号 |
| 2017-07-30 | 张元林 | 初始版本 | V1.0 |
| 2018-08-01 | 王绍成 | Java基础版本更新 | V2.0 |
| 2019-08-09 | 徐丽莎 | Java基础版本更新 | V3.0 |

**主讲人：**

**徐丽莎**

目录

[尚马教育 JAVA 基础课程 1](#_Toc25435)

[Lang包常用类 1](#_Toc16039)

[1. 包装类 2](#_Toc24937)

[1.1. 概念 2](#_Toc25446)

[1.2. 分类 3](#_Toc17942)

[1.3. Integer 3](#_Toc11713)

[1.3.1. 方法 3](#_Toc10876)

[1.3.2. 装箱与拆箱 3](#_Toc12357)

[1.3.3. 整数缓存池 3](#_Toc9575)

[1.4. Character 4](#_Toc828)

[1.4.1. 方法 4](#_Toc32767)

[1.4.2. 案例 4](#_Toc7461)

[2. Math 5](#_Toc18641)

[2.1. 方法 5](#_Toc17442)

[2.2. 案例 5](#_Toc31)

[3. Object 5](#_Toc1929)

[3.1. 方法 6](#_Toc12184)

[3.2. 案例 6](#_Toc27446)

[3.3. 克隆 clone 6](#_Toc4329)

[4. Class 6](#_Toc31324)

[4.1. 操作属性 7](#_Toc16841)

[4.2. 操作方法 7](#_Toc26082)

[4.3. 操作构造方法 7](#_Toc14698)

[5. 作业 7](#_Toc12936)

## 包装类

* java.lang包是java内置的一个基础包，其中包含了一系列程序中经常要用到的类；
* 在默认情况下，每个java程序都会自动导入该包，因此无需在程序中显式地声明。

### 概念

* 有基本数据类型为何又提供包装类呢？
  + 类似C++,java提供基本的数据类型将变量存储在栈里面，效率更加高效。
  + java是一门面向对象的语言，基本数据类型不具有对象的特质，为了让基本数据类型具有对象的特质，就出现了包装类。
  + 它相当于将基本类型“包装起来”，使得它具有了对象的性质，并且为其添加了属性和方法，丰富了基本类型的操作

### 分类

* 为了丰富基本数据类型(4类八种)的操作，提供8个包装类

|  |
| --- |
|  |

### Integer

* 以Integer为例，了解及掌握包装类相关的功能。
* Integer类包装一个对象中的原始类型int的值

#### 方法

|  |
| --- |
|  |

#### 装箱与拆箱

|  |
| --- |
|  |

#### 整数缓存池

|  |
| --- |
|  |

### Character

#### 方法

|  |
| --- |
|  |

#### 案例

|  |
| --- |
| **public** **class** CharacterDemo {  **public** **static** **void** main(String[] args) {  **char**[] charArray = { '\*', '7', 'b', ' ', 'A' };  **for** (**int** i = 0; i < charArray.length; i++) {  **if** (Character.*isDigit*(charArray[i])) {  System.***out***.println(charArray[i] + "是一个数字。");  }  **if** (Character.*isLetter*(charArray[i])) {  System.***out***.println(charArray[i] + "是一个字母。");  }  **if** (Character.*isWhitespace*(charArray[i])) {  System.***out***.println(charArray[i] + "是一个空格。");  }  **if** (Character.*isLowerCase*(charArray[i])) {  System.***out***.println(charArray[i] + "是小写形式。");  }  **if** (Character.*isUpperCase*(charArray[i])) {  System.***out***.println(charArray[i] + "是大写形式。");  }  }  }  } |

## Math

* Math类中提供了一系列基本数学运算和几何运算的方法；
* 该类的构造方法被修饰为private，因此不能实例化；
* 该类中的所有方法都是静态的，可以通过类名直接调用；
* 该类被修饰为final，因此没有子类

### 方法

|  |
| --- |
|  |

### 案例

|  |
| --- |
|  |

## Object

* Java中的类体系遵循单根结构，即任何一个类往上追溯都到达同一个父类；
* Object类就是这个单根体系的根，也就是说它是其它所有类的共同父类；
* 如果用户定义的类没有扩展任何其它类，则默认扩展自Object类；
* Object类中定义的一些方法，会被继承到所有类中

### 方法

|  |
| --- |
|  |

### 案例

|  |
| --- |
|  |

### 克隆 clone

|  |
| --- |
|  |

## Class(反射)

* Java应用程序实际上都是由一个个对象组成，这些对象分别属于什么类，是从哪个类继承而来，这一系列的信息都是由Class类的实例来记录的；
* Class类的实例用于记录对象的类描述信息；
* 如果在程序运行时，需要检索某个对象的相关类信息，可以调用该对象的getClass方法来获得记录其描述信息的Class类实例；
* Class类没有公共的构造方法，无法通过new运算符实例化，只能通过对象的getClass方法，或是通过Class的静态方法forName或者是class属性来获得实例。
* 反射： 通过Class的对象 操作类或者接口的属性 方法 构造
  + 在反射里面，将类的属性封装成Field进行操作
  + 将类的方法封装成Method进行操作
  + 将类的构造封装成Constructor进行操作

### 操作属性

|  |
| --- |
|  |

### 操作方法

|  |
| --- |
|  |

### 操作构造方法

|  |
| --- |
|  |

## 作业

1. 不利用Integer.parseInt()方法将一个给定的字符串转变为Int型数值,并检测自己实现的方法和Integer.parseInt()的性能差距。
2. 编写一个Student类，包含name、age等属性，要求使用System.out.println()打印Student类的对象引用时，输出的为name的值！